

LAPORAN TUGAS KECIL STRATEGI ALGORITMA
IMPLEMENTASI ALGORITMA UCS DAN A* UNTUK MENENTUKAN
LINTASAN TERPENDEK



Disusun Oleh:

Muhhamad Syauqi Jannatan **13521014**

M. Malik I. Baharsyah **13521029**

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

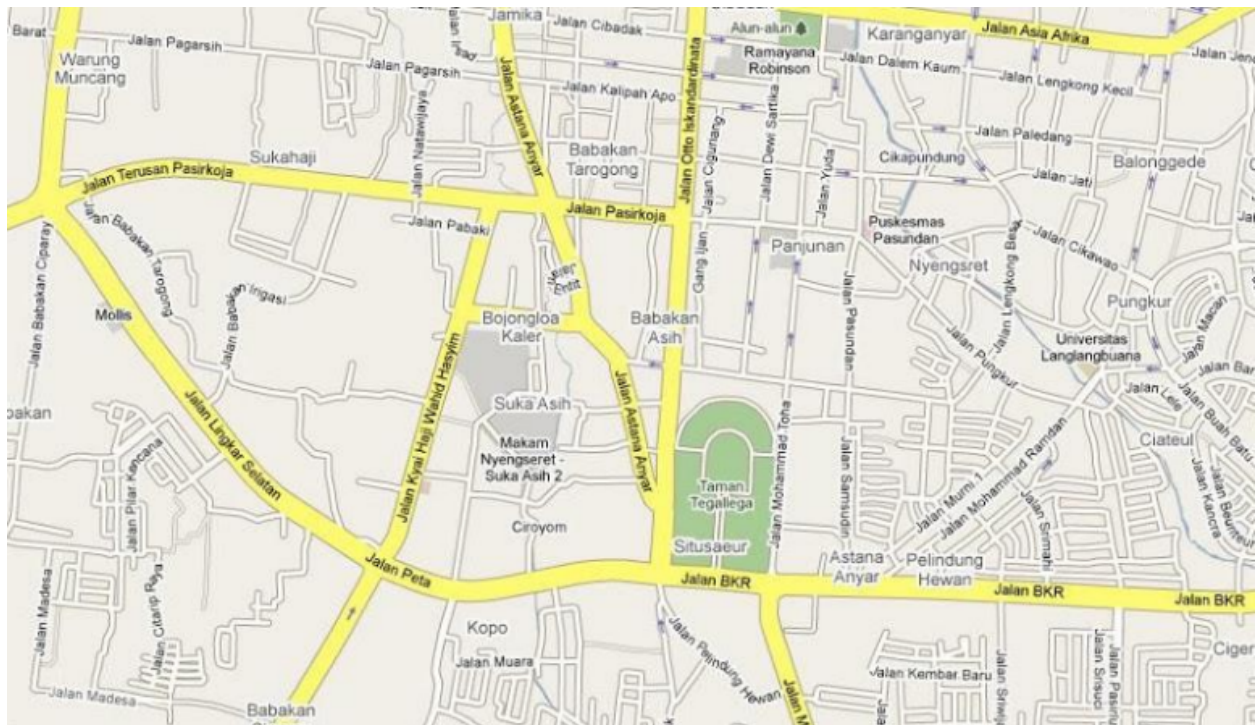
DAFTAR ISI

BAB I	3
DESKRIPSI PERSOALAN	3
BAB II	5
IMPLEMENTASI PROGRAM	5
2.1 Source Code Program	5
2.1.1 main.py	5
2.1.2 GUI.py	5
2.1.3 astar.py	10
2.1.4 ucs.py	11
2.1.5 util.py	13
2.2 Uji Coba	13
2.2.1 Peta di Sekitar ITB	13
2.2.2 Peta di Sekitar Alun-alun Bandung	15
2.2.3 Peta di Sekitar Bandung Selatan	16
2.2.4 Peta di Sekitar Jatinangor	18
2.2.5 File Test Case Error	19
BAB III	21
KESIMPULAN	21
LAMPIRAN	22

BAB I

DESKRIPSI PERSOALAN

Algoritma UCS (Uniform cost search) dan A* (atau A star) merupakan algoritma yang dapat digunakan untuk menentukan lintasan terpendek dari suatu titik ke titik lain. Pada tugas kecil 3 ini, program diminta untuk menentukan lintasan terpendek berdasarkan peta Google Map jalan-jalan di kota Bandung. Dari ruas-ruas jalan di peta dibentuk graf. Simpul menyatakan persilangan jalan (simpang 3, 4 atau 5) atau ujung jalan. Asumsi jalan dapat dilalui dari dua arah. Bobot graf menyatakan jarak (m atau km) antar simpul. Jarak antara dua simpul dapat dihitung dari koordinat kedua simpul menggunakan rumus jarak Euclidean (berdasarkan koordinat) atau dapat menggunakan ruler di Google Map, atau cara lainnya yang disediakan oleh Google Map.



Langkah pertama di dalam program ini adalah membuat graf yang merepresentasikan peta (di area tertentu, misalnya di sekitar Bandung Utara/Dago) dari input file berformat txt yang berisi jumlah simpul, nama masing-masing simpul, dan matriks ketetanggaan dengan representasi jarak dalam satuan meter. Nilai positif pada input matriks ketetanggaan menyatakan bahwa simpul tersebut terhubung/bertetangga dengan simpul yang lain dengan jarak yang dimaksud, sedangkan nilai negatif pada input matriks ketetanggaan menyatakan simpul tersebut tidak terhubung tetapi memiliki jarak yang

IF2211

Strategi Algoritma

dimaksud bila diukur menggunakan *ruler* di Google Maps. Berdasarkan graf yang dibentuk, program menerima input simpul asal dan simpul tujuan, lalu menentukan lintasan terpendek antara keduanya menggunakan algoritma UCS atau A* sesuai dengan pilihan pengguna. Lintasan terpendek dapat ditampilkan pada peta/graf (misalnya jalan-jalan yang menyatakan lintasan terpendek diberi warna merah). Nilai heuristik yang dipakai adalah jarak garis lurus dari suatu titik ke tujuan menggunakan bantuan *ruler* di Google Maps. Setelah mendapatkan semua input yang dibutuhkan untuk mencari nilai *cost*/jarak tempuh, pada akhirnya program akan menampilkan nilai *cost*/jarak tempuh tersebut beserta rute dari simpul awal hingga menuju ke simpul akhir dari masukan pengguna beserta plotting. Apabila rute untuk menuju ke simpul akhir tidak ditemukan, maka program tidak akan menampilkan nilai *cost*/jarak tempuh maupun rute, tetapi program tetap menampilkan plotting.

BAB II

IMPLEMENTASI PROGRAM

2.1 Source Code Program

2.1.1 main.py

```
import GUI

if __name__ == '__main__':
    GUI.main()
```

2.1.2 GUI.py

```
import os
import ast
import tkinter as tk
import customtkinter as ctk
import networkx as nx
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure
import matplotlib.pyplot as plt
import Algorithms.ucs as ucs
import Algorithms.astar as astar
import Algorithms.util as util

ctk.set_appearance_mode("System") # Modes: system (default), light, dark
ctk.set_default_color_theme("blue") # Themes: blue (default), dark-blue, green

plot = None
canvas_frame = None

def main():
    #create main window
    main_window = ctk.CTk()
    main_window.geometry("800x600")
    main_window.title("Path Finding")
```

IF2211

Strategi Algoritma

```
nodes = ctk.StringVar(value=[])
adj_matrix = ctk.StringVar(value=[[[]]])

#create the options frame
options_frame = ctk.CTkFrame(main_window)
options_frame.pack(padx=20, pady=20, side="left", fill="both")

#add text to the options frame with font size 12
ctk.CTkLabel(options_frame, text="Choose Algorithm", font=("Arial",
16)).pack(padx=10, pady=10)

#create variable to store selected option
selected_algorithm = ctk.StringVar()

# create option radio buttons
ucsButton = ctk.CTkRadioButton(options_frame, text="Uniform Cost
Search", variable=selected_algorithm, value="UCS")
aStarButton = ctk.CTkRadioButton(options_frame, text="A Star Search",
variable=selected_algorithm, value="A*")

#pack the option buttons
ucsButton.pack(anchor="w", padx=20, pady=10)
aStarButton.pack(anchor="w", padx=20, pady=10)

filename_label = ctk.CTkLabel(options_frame, text="Selected file:
None", font=("Arial", 14))
filename_label.pack(anchor="w", padx=20, pady=20)

# add open file button
openFileButton = ctk.CTkButton(options_frame, text="Open File",
command=lambda: openFileButton_clicked(filename_label, adj_matrix, nodes))
openFileButton.pack(anchor="w", padx=20, pady=10)

# add starting node label
startingNodeLabel = ctk.CTkLabel(options_frame, text="Choose Starting
Node", font=("Arial", 12))
startingNodeLabel.pack(anchor="w", padx=20, pady=5)

# add starting node dropdown menu
```

IF2211

Strategi Algoritma

```
starting_node = ctk.StringVar()
startingNodeDropdown = ctk.CTkOptionMenu(options_frame,
variable=starting_node, values=nodes.get())
startingNodeDropdown.pack(anchor="w", padx=20, pady=5)

# add starting node label
startingNodeLabel = ctk.CTkLabel(options_frame, text="Choose Finishing
Node", font=("Arial", 12))
startingNodeLabel.pack(anchor="w", padx=20, pady=5)

# add finishing node dropdown menu
finishing_node = ctk.StringVar()
finishingNodeDropdown = ctk.CTkOptionMenu(options_frame,
variable=finishing_node, values=nodes.get())
finishingNodeDropdown.pack(anchor="w", padx=20, pady=5)

# add Find Path button
findPathButton = ctk.CTkButton(options_frame, text="Find Path",
    command=lambda: findPathButton_clicked(selected_algorithm.get(),
starting_node.get(), finishing_node.get(), adj_matrix.get(), nodes.get()))
findPathButton.pack(anchor="w", padx=20, pady=20)

canvas_frame = ctk.CTkFrame(main_window, height=600)
canvas_frame.pack(padx=20, pady=20, fill="both", expand=True)

canvas = ctk.CTkCanvas(canvas_frame)
canvas.pack(fill="both", expand=True)

route_label = ctk.CTkLabel(canvas_frame, text="Route: ",
font=("Arial", 14))
route_label.pack(anchor="w", padx=10, pady=10)

cost_label = ctk.CTkLabel(canvas_frame, text="Cost: ", font=("Arial",
14))
cost_label.pack(anchor="w", padx=10, pady=10)

def openFileButton_clicked(filename_label, adj_matrix, nodes):
    if selected_algorithm.get() == "":
        show_popup("Please select an algorithm first")
```

IF2211

Strategi Algoritma

```
        else:
            file_path = open_file()
            if file_path != "":
                filename_label.configure(text="Selected file: " +
os.path.basename(file_path))
                graph = util.read_graph(file_path)
                nodes.set(graph[0])
                adj_matrix.set(graph[1])
                nodes_list = [item.strip('\\"') for item in
nodes.get().strip('()').split(', ')]
                startingNodeDropdown.configure(values=nodes_list)
                finishingNodeDropdown.configure(values=nodes_list)

    def findPathButton_clicked(algorithm, starting_node, finishing_node,
adj_matrix, nodes):
        if algorithm == "":
            show_popup("Please select an algorithm first")
        elif starting_node == "":
            show_popup("Please select a starting node")
        elif finishing_node == "":
            show_popup("Please select a finishing node")
        else:
            nodes_list = [item.strip('\\"') for item in
nodes.strip('()').split(', ')]
            adj_list = string_to_adj_matrix(adj_matrix)
            if algorithm == "UCS":
                result = ucs.ucs(adj_list,
nodes_list.index(starting_node), nodes_list.index(finishing_node),
nodes_list)
            else:
                result = astar.astar(adj_list,
nodes_list.index(starting_node), nodes_list.index(finishing_node),
nodes_list, astar.heuristic)
            add_graph(adj_list, result[1], nodes_list, result[0])

# use networkx to draw graph in a frame
def add_graph(adj_list, path, nodes, cost):
    global plot
    if plot:
```


IF2211

Strategi Algoritma

```
        plot.get_tk_widget().destroy()
        #clear the matplotlib figure
        plt.clf()
    G = nx.Graph()
    for i in range(len(adj_list)):
        for neighbor, weight in adj_list[i]:
            if weight > 0:
                G.add_edge(nodes[i], nodes[neighbor], weight=weight)

    node_colors = ['red' if i in path else 'blue' for node in
G.nodes()]
    edge_colors = ['red' if (u, v) in zip(path, path[1:]) or (v, u) in
zip(path, path[1:]) else 'black' for u, v in G.edges()]
    pos = nx.spring_layout(G)
    nx.draw_networkx(G, pos, node_color=node_colors,
edge_color=edge_colors, with_labels=True)
    labels = nx.get_edge_attributes(G, 'weight')
    nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)

    # show the plot on the canvas
    fig = plt.gcf()
    plot = FigureCanvasTkAgg(fig, master=canvas)
    plot.draw()
    plot.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH, expand=1)

    # add cost label
    if cost != None:
        cost_label.configure(text="Cost: "+str(cost))
        route_label.configure(text="Route: " + " -> ".join(path))

main_window.mainloop()

def create_canvas_frame(main_window):
    global canvas_frame
    canvas_frame = ctk.CTkFrame(main_window, height=600)
    canvas_frame.pack(padx=20, pady=20, fill="both", expand=True)

def open_file():
```

IF2211

Strategi Algoritma

```
file_path = ctk.filedialog.askopenfilename(
    title="Select a Text File",
    filetypes=[("Text Files", "*.txt")]
)
return file_path

def show_popup(message):
    tk.messagebox.showinfo("Popup Message", message)

def string_to_adj_matrix(adj_matrix_string):
    adj_matrix = ast.literal_eval(adj_matrix_string)
    adj_list = [list(t) for t in adj_matrix]
    return adj_list
```

2.1.3 astar.py

```
import heapq

def astar(adj_list, start, goal, nodes, heuristic):
    visited = set()
    heap = [(0, 0, start, [])]
    while heap:
        (estimated_cost, real_cost, current, path) = heapq.heappop(heap)
        if current in visited:
            continue
        visited.add(current)
        path = path + [current]
        if current == goal:
            path_nodes = [nodes[i] for i in path]
            return real_cost, path_nodes
        for neighbor, weight in adj_list[current]:
            if neighbor not in visited and weight > 0:
```

IF2211

Strategi Algoritma

```
        new_real_cost = real_cost + weight

        estimated_cost = new_real_cost + heuristic(neighbor, goal,
adj_list)

        heapq.heappush(heap, (estimated_cost, new_real_cost,
neighbor, path))

    return None, None

def heuristic(start, goal, adj_list):

    for neighbor, weight in adj_list[start]:

        if neighbor == goal:

            return abs(weight)

    return 0
```

2.1.4 ucs.py

```
import heapq

import matplotlib.pyplot as plt

import networkx as nx

def ucs(adj_list, start, goal, nodes):

    visited = set()

    heap = [(0, start, [])]

    while heap:

        (cost, current, path) = heapq.heappop(heap)

        if current in visited:

            continue

        visited.add(current)

        path = path + [current]
```

IF2211

Strategi Algoritma

```
        if current == goal:

            path_nodes = [nodes[i] for i in path]

            return cost, path_nodes

        for neighbor, weight in adj_list[current]:

            if neighbor not in visited and weight > 0:

                heapq.heappush(heap, (cost + weight, neighbor, path))

    return None, None

def show_graph(adj_list, path, nodes):

    G = nx.Graph()

    for i in range(len(adj_list)):

        for neighbor, weight in adj_list[i]:

            G.add_edge(nodes[i], nodes[neighbor], weight=weight)

    node_colors = ['red' if i in path else 'blue' for node in G.nodes()]

    edge_colors = ['red' if (u, v) in zip(path, path[1:]) or (v, u) in
zip(path, path[1:]) else 'black' for u, v in G.edges()]

    pos = nx.spring_layout(G)

    nx.draw_networkx(G, pos, node_color=node_colors,
edge_color=edge_colors, with_labels=True)

    labels = nx.get_edge_attributes(G, 'weight')

    nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)

    plt.show()
```

2.1.5 util.py

```
def read_graph(file_name):  
    with open(file_name, 'r') as f:  
        n = int(f.readline().strip())  
        nodes = f.readline().strip().split(',')  
        adj_list = [[] for i in range(n)]  
        for i in range(n):  
            row = list(map(float, f.readline().strip().split(',')))  
            for j, w in enumerate(row):  
                if w != 0:  
                    adj_list[i].append((j, w))  
                    adj_list[j].append((i, w))  
        return nodes, adj_list
```

2.2 Uji Coba

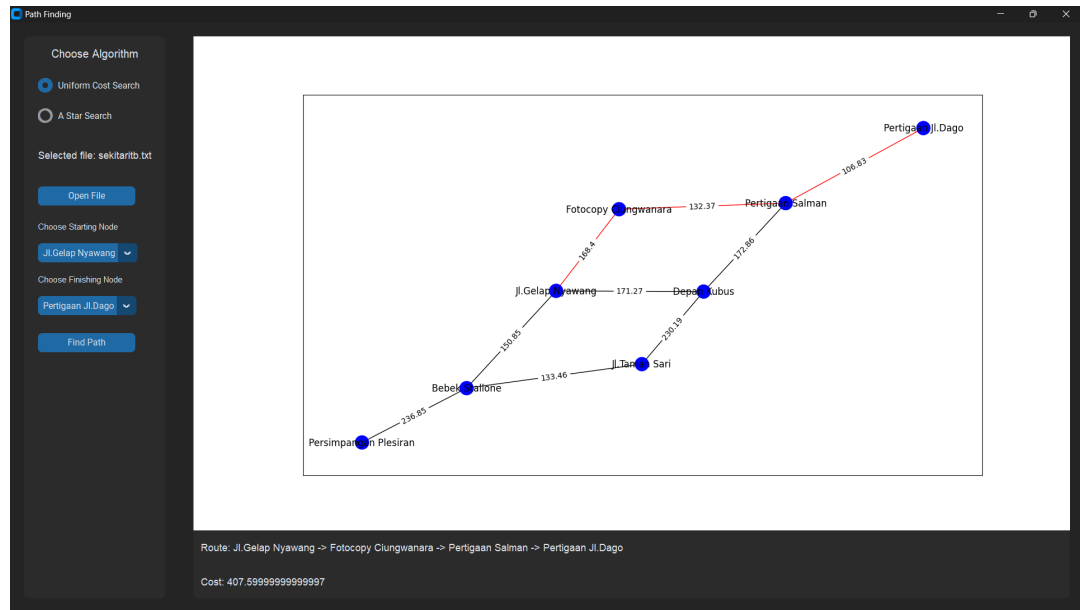
2.2.1 Peta di Sekitar ITB

Input	8 Jl.Taman Sari,Depan Kubus,Pertigaan Salman,Pertigaan Jl.Dago,Bebek Stallone,Jl.Gelap Nyawang,Fotocopy Ciungwanara,Persimpangan Plesiran 0, 230.19, -390.16, -494.77, 133.46, -222.16, -376.76, -369.83 230.19, 0, 172.86, -311.55, -234.08, 171.27, -239.54, -406.78 -390.16, 172.86, 0, 106.83, -378.00, -237.34, 132.37, -444.38 -494.77, -311.55, 106.83, 0, -471.24, -326.87, -175.63, -500.11 133.46, -234.08, -378.00, -471.24, 0, 150.85, -320.15, 236.85 -222.16, 171.27, -237.34, -326.87, 150.85, 0, 168.40, -234.69 -376.76, -239.54, 132.37, -175.63, -320.15, 168.40, 0, -325.40 -369.83, -406.78, -444.38, -500.11, 236.85, -234.69, -325.40, 0
Output	Algoritma UCS

IF2211

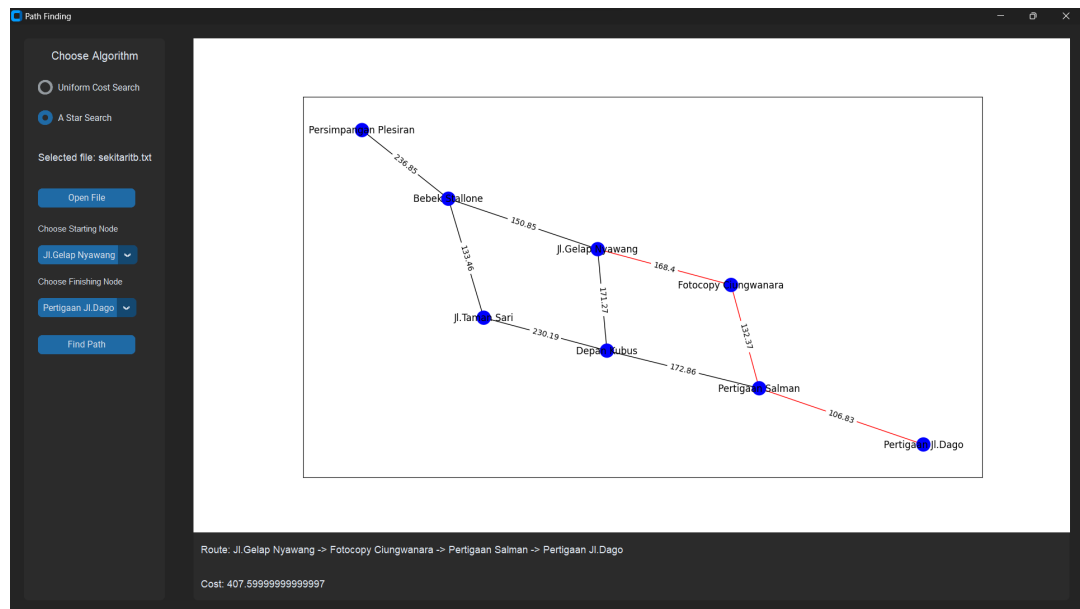
Strategi Algoritma

Simpul awal: Jl.Gelap Nyawang
Simpul akhir: Pertigaan Jl.Dago

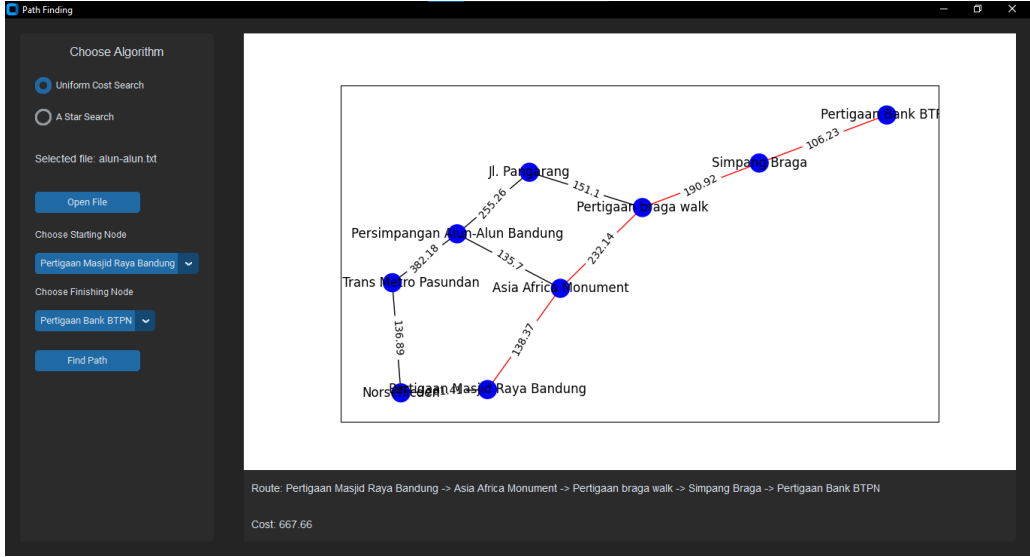


Algoritma A*

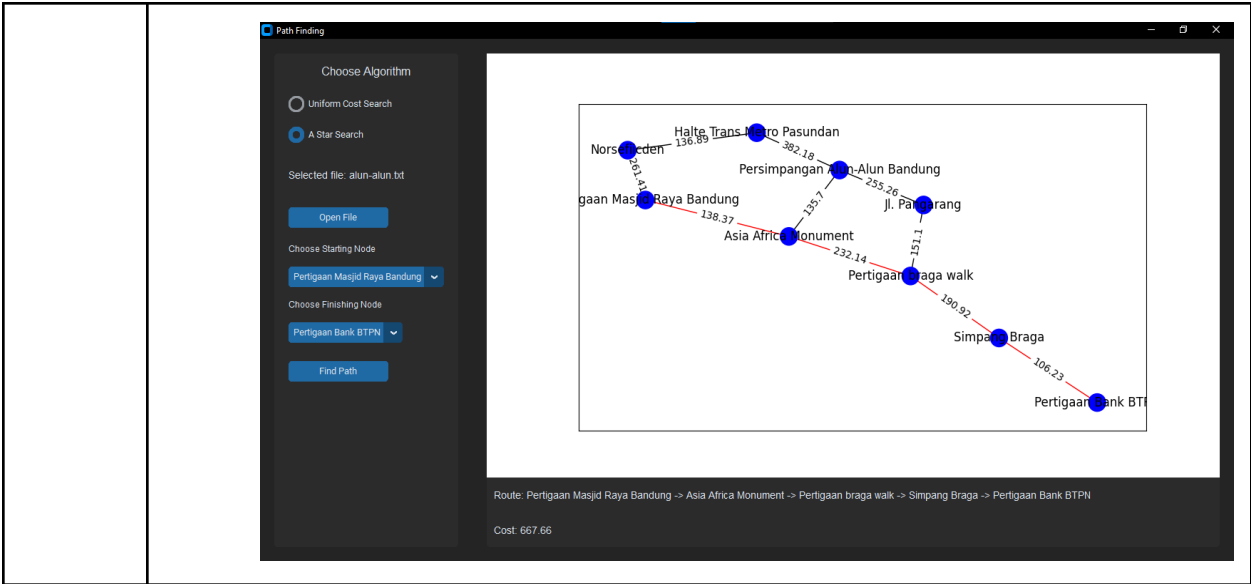
Simpul awal: Jl.Gelap Nyawang
Simpul akhir: Pertigaan Jl.Dago



2.2.2 Peta di Sekitar Alun-alun Bandung

<p>Input</p>	<p>9</p> <p>Norsefiicden,Pertigaan Masjid Raya Bandung,Asia Africa Monument,Pertigaan braga walk,Simpang Braga,Pertigaan Bank BTPN,Jl. Pangarang,Persimpangan Alun-Alun Bandung,Halte Trans Metro Pasundan</p> <p>0, 261.41, -400.23, -634.56, -655.27, -553.58, -664.04, -421.32, 134.67 261.41, 0, 138.37, -370.59, -410.86, -314.40, -413.19, -198.12, -291.34 -400.23, 138.37, 0, 232.14, -303.24, -223.90, -284.91, 135.70, -416.14 -634.56, -370.59, 232.14, 0, 190.92, -209.65, 151.10, -281.30, -641.73 -655.27, -410.86, -303.24, 190.92, 0, 106.23, -340.95, -413.00, -704.74 -553.58, -314.40, -223.90, -209.65, 106.23, 0, -351.38, -350.38, -606.82 -664.04, -413.19, -284.91, 151.10, -340.95, -351.38, 0, 255.26, -643.91 -421.32, -198.12, 135.70, -281.30, -413.00, -350.38, 255.26, 0, 382.18</p>
<p>Output</p>	<p>Algoritma UCS</p> <p>Simpul awal: Pertigaan Masjid Raya Bandung</p> <p>Simpul akhir: Pertigaan Bank BTPN</p>  <p>Algoritma A*</p> <p>Simpul awal: Pertigaan Masjid Raya Bandung</p> <p>Simpul akhir: Pertigaan Bank BTPN</p>

IF2211
Strategi Algoritma

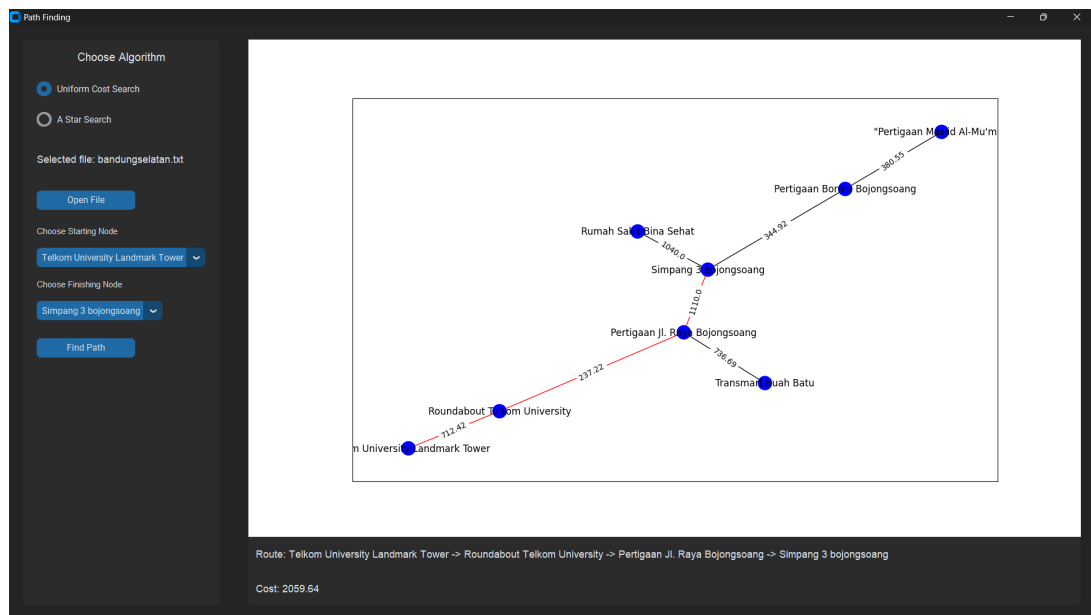


2.2.3 Peta di Sekitar Bandung Selatan

Input	8 Telkom University Landmark Tower, Roundabout Telkom University, Pertigaan Jl. Raya Bojongsoang, Transmart Buah Batu, Simpang 3 Bojongsoang, Pertigaan Borma Bojongsoang, Rumah Sakit Bina Sehat, Pertigaan Masjid Al-Mu'min 0, 712.42, -935.29, -1050.00, -1600.00, -1890.00, -1930.00, -2060.00 712.42, 0, 237.22, -747.29, -1120.00, -1450.00, -1830.00, -1510.00 -935.29, 237.22, 0, 736.69, 1110.00, -1450.00, -1930.00, -1450.00 -1050.00, -747.29, 736.69, 0, -1810.00, -2140.00, -2560.00, -2130.00 -1600.00, -1120.00, 1110.00, -1810.00, 0, 344.92, 1040.00, -480.11 -1890.00, -1450.00, -1450.00, -2140.00, 344.92, 0, -869.14, 380.55 -1930.00, -1830.00, -1930.00, -2560.00, 1040.00, -869.14, 0, -1250.00 -2060.00, -1510.00, -1450.00, -2130.00, -480.11, 380.55, -1250.00, 0
Output	Algoritma UCS Simpul awal: Telkom University Landmark Tower Simpul akhir: Simpang 3 Bojongsoang

IF2211

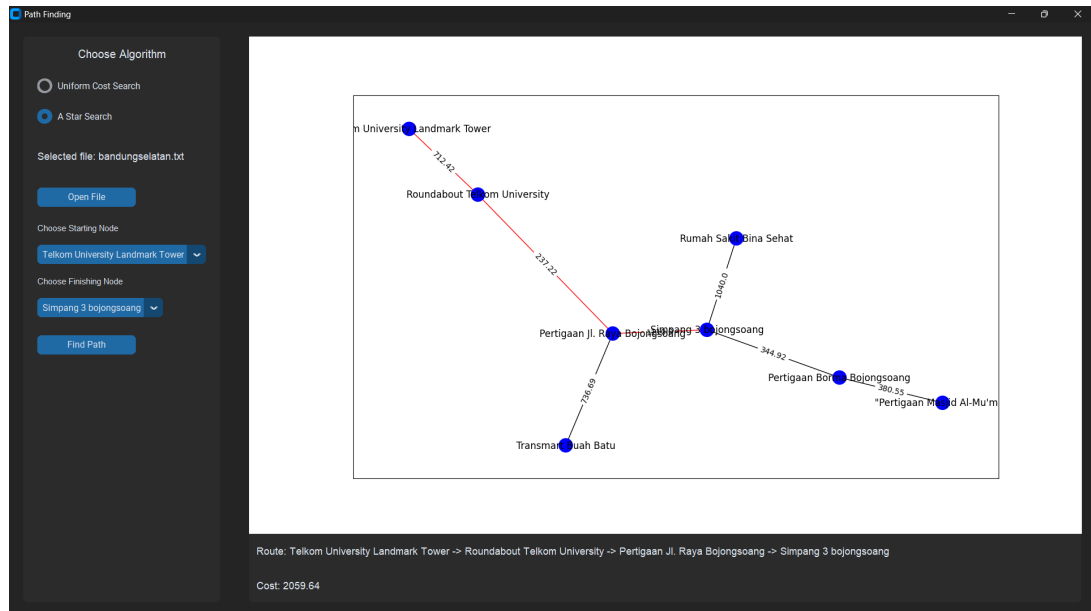
Strategi Algoritma



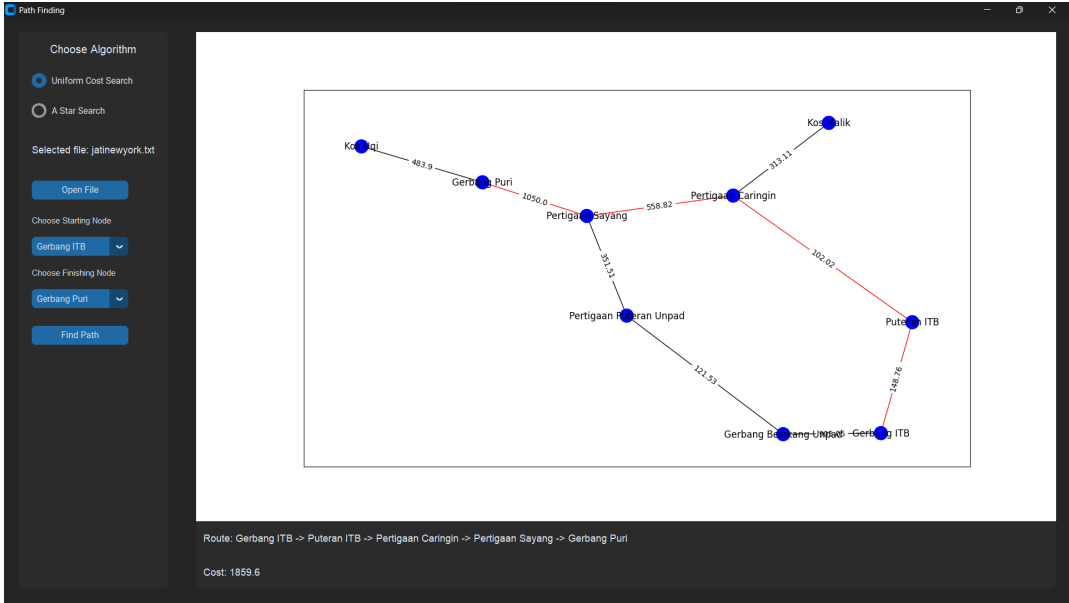
Algoritma A*

Simpul awal: Telkom University Landmark Tower

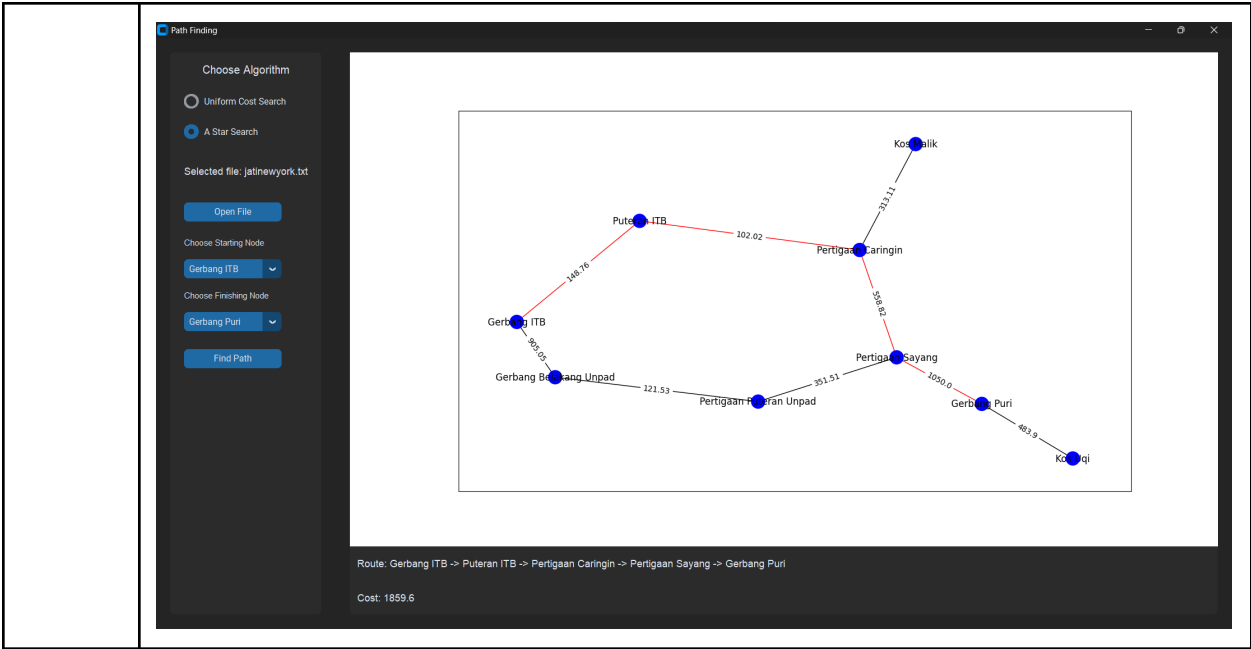
Simpul akhir: Simpang 3 Bojongsoang



2.2.4 Peta di Sekitar Jatinangor

<p>Input</p>	<p>9</p> <p>Puteran ITB, Gerbang ITB, Gerbang Belakang Unpad, Pertigaan Puteran Unpad, Pertigaan Sayang, Gerbang Puri, Kos Uqi, Pertigaan Caringin, Kos Malik</p> <p>0, 148.76, -1030.00, -1020.00, -667.35, -1100.00, -1060.00, 102.02, -321.04</p> <p>148.76, 0, 905.05, -895.59, -546.36, -1150.00, -1040.00, -103.47, -385.61</p> <p>-1030.00, 905.05, 0, 121.53, -372.33, -1280.00, -853.95, -924.47, -1000.00</p> <p>-1020.00, -895.59, 121.53, 0, 351.51, -1180.00, -751.20, -908.05, -956.14</p> <p>-667.35, -546.36, -372.33, 351.51, 0, 1050.00, -724.24, 558.82, -634.54</p> <p>-1100.00, -1150.00, -1280.00, -1180.00, 1050.00, 0, 483.90, -1050.00, -773.46</p> <p>-1060.00, -1040.00, -853.95, -751.20, -724.24, 483.90, 0, -978.27, -795.95</p> <p>102.02, -103.47, -924.47, -908.05, 558.82, -1050.00, -978.27, 0, 313.11</p> <p>-321.04, -385.61, -1000.00, -956.14, -634.54, -773.46, -795.95, 313.11, 0</p>
<p>Output</p>	<p>Algoritma UCS</p> <p>Simpul awal: Gerbang ITB</p> <p>Simpul akhir: Gerbang Puri</p>  <p>Algoritma A*</p> <p>Simpul awal: Gerbang ITB</p> <p>Simpul akhir: Gerbang Puri</p>

IF2211
Strategi Algoritma



2.2.5 File Test Case Error

Input	3 Jl.Taman Sari,Depan Kubus,Pertigaan Jl.Dago 0, 230.19, -486.79 230.19, 0, -272.24 -486.79, -272.24, 0
Output	<p>The screenshot shows the 'Path Finding' application with 'Uniform Cost Search' selected. A 'Popup Message' dialog box is displayed in the center with the text 'No path found' and an 'OK' button. The main area is empty, and the route and cost fields at the bottom are also empty.</p>

BAB III

KESIMPULAN

Algoritma UCS (Uniform Cost Search) dan A* (A-star) dikenal sebagai algoritma pencarian lintasan terpendek yang sering digunakan dalam masalah optimasi jalur. Meskipun keduanya berbeda dalam menggunakan heuristik, keduanya dapat mencapai tujuan yang sama yaitu menemukan jalur terpendek antara dua titik. Penggunaan algoritma A* dengan nilai heuristik yang dipakai adalah jarak garis lurus dari suatu titik ke titik tujuan membuat penelusuran rute menjadi lebih efisien secara garis besar karena fungsi heuristik tersebut menuntun pencarian menuju goal state dengan perkiraan cost seminimal mungkin. Sedangkan Algoritma UCS tidak menggunakan heuristik. Algoritma UCS mengeksplorasi semua jalur yang mungkin secara sistematis dan memilih jalur dengan cost terkecil pada setiap tahap. Perlu diingat bahwa algoritma A* hanya akan menghasilkan solusi yang lebih optimal jika nilai heuristik yang digunakan akurat dan diterapkan dengan benar. Berdasarkan hasil pengujian kami, penggunaan algoritma UCS dan A* untuk kasus-kasus di atas akan menuntun kepada rute solusi yang sama.

LAMPIRAN

Pranala Github : https://github.com/malikhaharsyah/Tucil3_13521014_13521029

Checklist :

Poin	Ya	Tidak
1. Program dapat menerima input graf	✓	
2. Program dapat menghitung lintasan terpendek dengan UCS	✓	
3. Program dapat menghitung lintasan terpendek dengan A*	✓	
4. Program dapat menampilkan lintasan terpendek serta jaraknya	✓	
5. Bonus: Program dapat menerima input peta dengan Google Map API dan menampilkan peta serta lintasan terpendek pada peta		✓